CLASSIFICATION OF FUTURISTIC TECHNOLOGIES DESCRIBED IN SPECULATIVE FICTION NOVELS

An Undergraduate Research Scholars Thesis

by

KEVIN SITTSER

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Ruihong Huang

May 2020

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

Classification of Futuristic Technologies Described in Speculative Fiction Novels

Kevin Sittser
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Ruihong Huang
Department of Computer Science
Texas A&M University

Speculative fiction works of literature are full of unreal futuristic technologies meant to amuse and interest the reader in a potential or alternate future. Some readings of such novels occasionally reveal that through time and the progression of technology in the real world, we have achieved some of the ideas put forth in these works of fiction. To properly prepare for a world of changing and developing technology, it is of great interest to use these fictional ideas to potentially predict the advancement of technology in real life. This project aims to create a machine learning system to analyze text passages introducing such futuristic technologies and classify them into categories related to their features and usage.

SECTION I

INTRODUCTION

In recent years, machine text classification has become increasingly useful as computers are used to make faster, more efficient analysis of text. Today's parsers can identify sentence structures with upwards of 80% accuracy and can often create reasonable summaries of texts they are given [1]. The purpose of this project is to create a machine learning system to read book excerpts dealing with fictional futuristic technology and classify these technologies into categories relating to their purposes and uses. These texts are obtained from the database Technovelgy, which contains several thousand descriptions of these futuristic ideas as well as an appropriate category for each excerpt. We utilize these texts to train our classifier to recognize elements of these labeled texts; our goal is that, given other technology descriptions from outside of this training set, our classifier should be able to reliably find an appropriate category for each passage. Ultimately, in a continuing project outside the scope of this paper, our classifier will be used to gather new information about ideas explained in these novels so that we can predict to some degree how soon the technologies described could actually be created in real life.

SECTION II

METHODS

Our system is written in Python. We primarily utilize the *scikit-learn* library available in Python to perform many of the techniques in this project [2].

1. Dataset

Our dataset consists of the data from Technovelgy's database, which contains 2997 passages from a variety of science fiction novels [3]. These passages all mention and describe a futuristic technology and categorize it into one of 31 labels such as "Robotics," "Weapon," or "Communication." Passages range from 4 to 1077 words, and there is a vastly different number of passages in each category, ranging from 4 to 488 passages.

We base our classifier system heavily on the work of on that of Miguel Fernández Zafra, who has published results of similar Python classification project online that achieved over 90% accuracy in classifying news articles by topic [4].

2. Preprocessing

To ensure that trivial features such as capitalizations and word conjugations are not treated as distinct features, we preprocess each passage using a series of common steps. We use NLTK's *word_tokenize* feature to deconstruct passages into individual word-based tokens and remove the stop words listed in the set of stop words provided by NLTK. Finally, we use the Porter Stemmer to stem the words, removing most affixes and inflections. (We also experimented with using WordNet's lemmatizer in place of the stemmer, but Porter Stemmer was able to more consistently remove some common suffixes that the lemmatizer did not.) An example of our preprocessing results is shown in Figure 1.

He spent several hours each day in the OOB's workshop, often with Blueshell. Reality graphics were a new thing to him, but after a few weeks he was beyond toy prototypes. The pressure suits he built had power packs and weapons stores.	→→→	spent sever hour day oob workshop often blueshel realiti graphic new thing week beyond toy prototyp pressur suit built power pack weapon store

Figure 1. Preprocessing a passage

3. Feature Extraction

We experiment with several different methods of feature extraction to identify features to train our classifier on. The most basic is *CountVectorizer*, which simply gathers a list of all unique tokens in the entire corpus and gives a binary label of the presence or absence of each token in each passage. An alternative mode of *CountVectorizer* counts the number of occurrences of each word. We test these two methods, as well as a more complex method, *TfidfVectorizer*. TF-IDF vectors score the importance of a word based on both the individual passage and the entire corpus, penalizing the importance if the word appears in a large proportion of the documents. Such words cannot be easily used to distinguish categories from each other, since they appear in documents from many categories.

We use a n-gram model included in these feature extractors and include bigrams and trigrams in case there are pairs or triplets of consecutive tokens (e.g., mentioning an object that has a two-word name), though we do not expect there to be a substantial number of important n-grams in the texts. As it turns out, this method makes minimal to no difference in the accuracy of our classifiers, as compared examining only individual words. However, all of our tests listed below include bigrams and trigrams that the feature extractors have deemed important.

4. Classification

We use 8-fold cross validation to train and test our classifiers. This method performs eight separate training and testing sequences, each performing testing on a separate one-eighth of

the data and training on the rest, ensuring large training sets of over 2600 passages while still performing testing on every single passage.

4.1 Naive Bayes Classifiers

Our initial attempt at categorization utilizes the Naive Bayes algorithm, as a simple baseline classification system. This popular method examines the value assigned to each feature in a test sample and compares training samples that share each of the same values, ultimately summing up a probability that each category is the most appropriate. Naive Bayes is relatively simple to implement and serves as an excellent starting point for our classification attempts. We use four different Naive Bayes classifiers provided in *scikit-learn*, namely *MultinomialNB*, *BernoulliNB*, *GaussianNB*, and *ComplementNB*. These classifiers are all optimized for slightly different data distributions and feature types, with *MultinomialNB* set up for integer-valued features, *BernoulliNB* for binary features, *GaussianNB* for features that follow a Gaussian distribution throughout the data, and *ComplementNB* for datasets with differing amounts of training data for each class [5]. Considering the substantial imbalance in our dataset, *ComplementNB* is likely to fit our data best, with *MultinomialNB* and *BernoulliNB* perhaps also working well with the aforementioned token-count and token-binary feature encodings respectively.

For each of these Naive Bayes classifiers, we collect for each of the 31 categories the average accuracy, precision, recall, and F1 scores among the eight tests done in cross-validation. This will ensure a comprehensive evaluation of the success of each classifier.

4.2 Support Vector Classifiers

We next use support vector classifiers (SVCs) to attempt to classify the data. These classifiers visualize each text passage as a vector in multidimensional space, with each feature

representing a dimension. It then attempts to reconfigure these vectors into higher dimensions via a "kernel" function so that it can linearly separate vectors of each class from each other, creating distinct regions in the multidimensional space for each class. *scikit-learn* implements several different SVC classifiers with slightly different methods. In this project we test the *SVC*, *LinearSVC*, and *NuSVC* classifiers in this package.

4.2.1 Parameter Optimization

Unlike the Naive Bayes classifiers, the SVC classifiers include a number of parameters that hugely influence how the classifiers act. We experiment with several parameters available in *scikit-learn*'s implementation, adjusting in particular the type and shape of the kernel function. Based on the aforementioned work of Zafra, we use *scikit-learn*'s *RandomizedSearchCV* functionality to test many different combinations of values for these parameters. *RandomizedSearchCV* finds the combination of parameters that gives the highest accuracy for the classifier, as well as an estimation of what that accuracy is. We use the parameters yielded by this function in our SVC classifications.

4.3 Other Classifiers

In the interest of thorough testing, we try a few more miscellaneous classifier models. *SGDClassifier* and *LogisticRegression* respectively use stochastic gradient descent and logistic regression as optimization functions to minimize incorrect labelings in training. *KNeighborsClassifier* treats data as vectors in a similar manner to SVC and assigns categories based on the categories of the K training datapoints (we have chosen K=5) closest to it. *GradientBoostingClassifier* is an ensemble method that creates several decision trees to classify the data and combines them into one cohesive classifier. Finally, *MLPClassifier* implements Multilayer Perceptron, a feed-forward neural network particularly suited for data that is difficult

to separate linearly. Because of the diversity of our data and the complex multilayer nature of MLP, We expect this classifier to work particularly robustly for this project and do a better job of recognizing patterns in the data than many of the other classifiers. As for the SVC classifiers, we optimize a number of parameters of these classifiers using *RandomizedSearchCV*.

5. Complications

One significant complication we face in our classification is that many categories are similar to each other, and as a result, many of the passages from Technovelgy do not intuitively belong to only a single category. This is because Technovelgy is designed as a search engine for passages in its database, rather than a strict classification system of mutually exclusive categories. While Technovelgy has labeled each passage as belonging to a particular category, it can be clearly seen that the categorization for many passages is actually rather ambiguous. For instance, Technovelgy's categories include both "Weapon" and "Warfare" and both "Spacecraft" and "Vehicle," two pairs of categories that are likely to have substantial overlap between them. Figure 2 shows a passage that could plausibly belong to any of the "Machine," "Weapon," or "Warfare" categories, of which "Machine" has been selected.


```
"Gravity-Polarized Explosive (TDX)": {
     "description": "A chemical explosive that acts at an angle to the local gravitational
     field.",
     "link": "content.asp?Bnum=2460",
     "excerpt": "At the same moment, a lurid scarlet glare splashed over his face and the front
     of his suit, and red lances of light checkered the street. There was an almost-simultaneous
     flat crash, without weight in the thin air, but ugly-sounding.\r\n\r\n\"TDX!\" Anderson
     shouted, involuntarily.\r\n\n",
     "articles": {},
     "category": "Material"
},
```


Figure 2. The chemical explosive described in this passage could plausibly be any of Weapon, Warfare, or Material

To simplify the classification scheme, we plan to merge similar categories together so that the classifier will not be forced to distinguish between them. Ideally, we will ultimately have eight to ten categories, which will likely make our classifier much more reliable. However, we have not yet produced a list of merged categories that successfully improves accuracy, so this topic will require further research.

Another key complication in our data is that there are substantially different numbers of passages from each category. Figure 3 shows this disparity of category sizes. Because of this, classifiers tend to overfit the data by categorizing many passages as "Space Tech," which has by far the most data, when they should be of different classes. To combat this overfitting, we attempt several methods of reducing the effects of the different amounts of data.

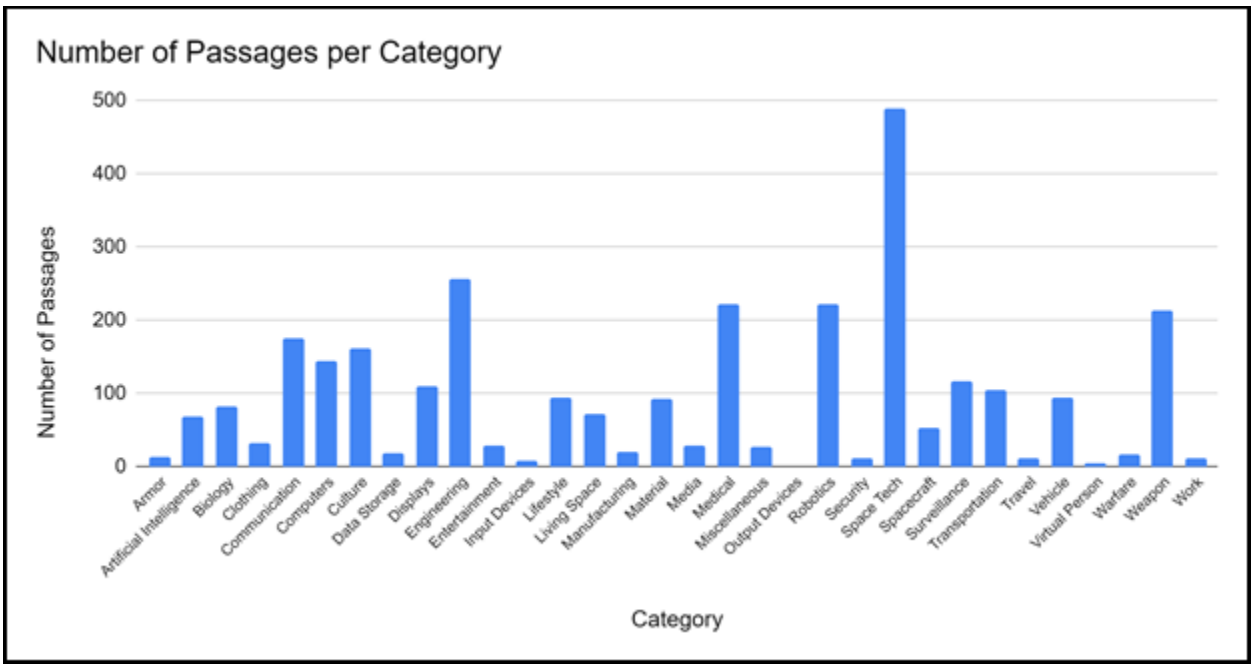

Figure 3. Categories have very different numbers of passages in them

We try normalizing the numbers of passages in each category. This method undersamples majority categories by deleting random passages from the training data, and oversamples minority categories by duplicating random passages. In this way we give each category an equal

number of passages to train on. While this method is imperfect, tending to overfit the oversampled minority classes and lose valuable information provided by the undersampled majority classes, it is a very popular method of dealing with imbalanced data and has proved to be reasonably effective [6]. For this project, we try normalizing category sizes to the 50th, 75th, and 95th percentiles of the sizes in our data.

Another strategy we use is to simply omit categories that are excessively large or small, training and testing on only data from categories that are of similar sizes. This should significantly improve accuracy, particularly since categories with a very small amount of data, which will likely achieve poor testing accuracy, are removed. While this does not accomplish our goal of building a classifier based on the entire corpus, it gives a better understanding of the maximum capabilities of our system and will serve as a baseline measure of success when we merge classes in the future.

SECTION III

RESULTS

Our best average accuracy score for our classifier over all 31 categories is 33.1%. This is not an ideal result, but some modification of the data, as listed below, gives some more promising results. We calculate both average overall accuracy (i.e., the proportion of all test samples that the classifier labeled correctly) and the average of each category's F1 score. F1 score is calculated based on both precision and recall and is generally a better descriptor of success for an imbalanced dataset such as ours, as it places more emphasis on false positives and false negatives found for each category. Like accuracy, F1 ranges from 0 (poor) to 1 (good).

1. Category Size Normalization

As expected, the most effective way to reduce the effects of class imbalance is to limit our classifier to a small number of large, similarly-sized classes. We choose the Engineering, Medical, Robotics, and Weapon categories, which are among the largest categories and ranged from 210 to 260 passages. This method of category size normalization increases accuracy to over 60% and F1 score to over 0.60 for most classifiers. Our best result is for *LinearSVC*, which achieves 66.1% accuracy and 0.660 F1 score. This result is much better, though it still could not be considered trustworthy overall in classifying a brand-new set of passages.

Undersampling and oversampling perform unexpectedly poorly; in fact, they reduce accuracy by several percentage points (to 31.3% in the case of the 33.1% score mentioned above). This method does not appear to be an effective way to normalize category sizes for this data.

2. Classifiers

Under our four-category setup, many of the classifiers perform very similarly well. Figure 4 gives the highest accuracy and F1 scores found for each of the twelve classifiers, out of our three different methods of feature extraction (binary labels of feature presence, numerical counts of features, and TF-IDF vectors). *LinearSVC*, as mentioned above, performs best in both accuracy (66.1%) and F1 score (0.660), and *LogisticRegression* performs only slightly less well (63.8%, 0.637). *GaussianNB* and *KNeighbors* perform worse than the rest of our classifiers by a substantial margin; it appears that our feature values are not on a Gaussian distribution as *GaussianNB* requires, nor do they form orderly enough vectors to be described by such a simplistic model as *KNeighbors*.

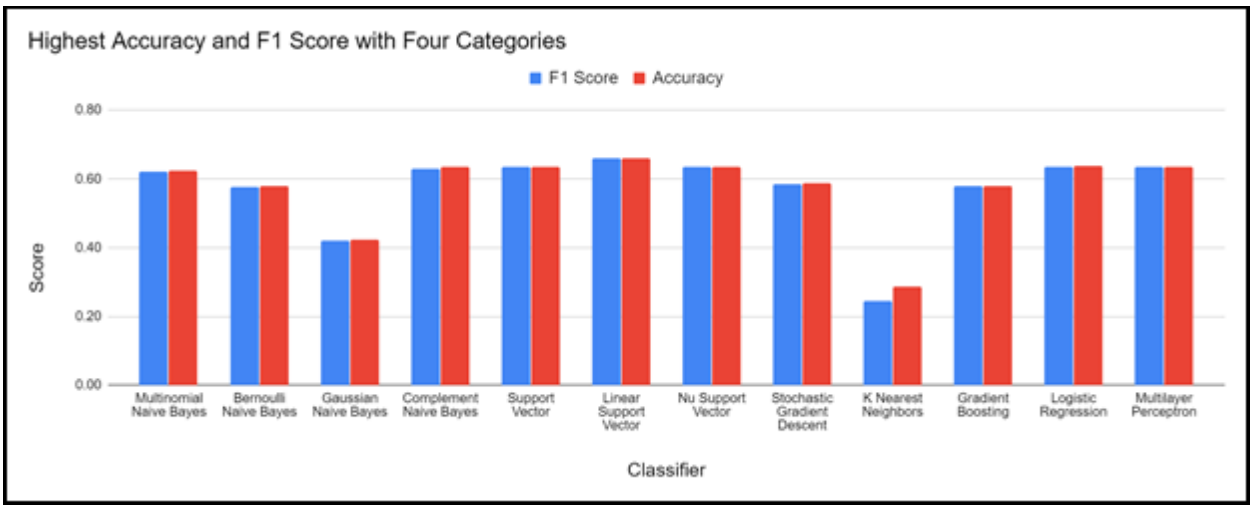

Figure 4. Accuracy and F1 scores for four similarly-sized categories

The accuracy and F1 scores for the best performing categories, when classifiers train on all 31 categories, for *LinearSVC* are listed in Table 1 and for *LogisticRegression* in Table 2. Unsurprisingly, they are mostly the largest categories, with the very largest category “Space Tech” having the greatest accuracy, and three of the four next largest having the next-greatest

accuracy scores. A confusion matrix reveals that the other large category, “Engineering,” very frequently has its passages identified as “Space Tech,” which gives it much less success.

We note that almost half of the categories, mostly the smallest ones, have 0% accuracy. This is unsurprising, as many of the categories have under twenty passages, so that the classifier simply is unable to learn enough information about them. One notable exception is the “Spacecraft” category, which has 52 passages but achieves 0% accuracy in both of these classifiers. The vast majority of “Spacecraft” passages are classified as “Space Tech,” indicating that these categories are too similar for the classifier to distinguish.

Table 1. Top category accuracy scores for LSVC classifier

Category	**Accuracy (%)**	**F1 Score**
Space Tech	87.1	0.501
Robotics	58.8	0.506
Weapon	55.4	0.510
Medical	46.4	0.373
Communication	32.0	0.310

Table 2. Top category accuracy scores for LR classifier

Category	**Accuracy (%)**	**F1 Score**
Space Tech	85.2	0.489
Robotics	60.2	0.477
Weapon	51.2	0.482
Medical	47.3	0.362
Communication	29.7	0.314

3. Feature Extraction

Results of the three feature extraction techniques vary by classifier, but TF-IDF tends to outperform the other two. Comparisons of accuracy and F1 scores for *LinearSVC* and *LogisticRegression* are listed in Tables 3 and 4, based on testing all 31 categories. Modeling with bigram and trigram features in addition to individual tokens had virtually no effect on accuracy.

Table 3. Accuracy and F1 Scores for LSVC classifier

Technique	**Accuracy (%)**	**F1 Score**
Count Vectors	28.0	0.139
Binary Vectors	27.9	0.134
TF-IDF Vectors	34.6	0.135

Table 4. Accuracy and F1 Scores for LR classifier

Technique	**Accuracy (%)**	**F1 Score**
Count Vectors	30.6	0.142
Binary Vectors	30.3	0.139
TF-IDF Vectors	33.1	0.120

SECTION IV

CONCLUSION

While our classifiers are not able to label passages with the full set of 31 categories with any reliability, they achieve better than random chance and therefore are somewhat successful. Further research should be done in several areas.

First of all, categories must be given approximately equal amounts of training data. There are a number of methods of doing this that we have not tried, and in order to successfully classify passages from the smaller categories, they will need to somehow be given more data.

Categories must be made more distinct from each other. Confusion matrices reveal substantial difficulty in distinguishing several classes that are intuitively similar. Perhaps the best way to solve both of these problems is to merge categories together, in particular merging small categories into larger ones, so that the classifier both has enough data in each category to correctly recognize features from the data and can make a clear distinction between categories. Further research is needed to create a merging scheme that creates clearly different categories and also results in super-categories that still represent a clear function or feature of the technologies therein.

Further research is also perhaps needed into feature encoding methods. Word embedding schemes such as GloVe and BERT are designed to capture elements of meanings of words so that more information can be gathered about the usage of each token and its similarity to other words. Such methods would be able to give the classifier a deeper knowledge of each passage and perhaps give it a better understanding of concepts that are similar between passages.

Finally, exploration of better classifiers is needed. It is evident from our extensive testing that the classifiers available in *scikit-learn* are not sufficient to reliably model the data, even when category sizes are normalized, as we accomplished with using only four similarly-sized categories. Deep neural networks like LSTMs and CNNs have been shown by various research to be very successful in text classification problems, and such classifiers could likely improve our results.

REFERENCES

[1] Xun Wang, Yasuhisa Yoshida, Tsutomu Hirao, Xun Wang, Yasuhisa Yoshida, Tsutomu Hirao, Katsuhito Sudoh, Masaaki Nagata, Katsuhito Sudoh, and Masaaki Nagata. 2015. Summarization Based on Task-Oriented Discourse Parsing. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23, 8 (Aug. 2015), 1358-1367. DOI: 10.1109/taslp.2015.2432573

[2] Fabian Pedregosa. 2011. Scikit-learn: Machine Learning in Python. *JMLR 12*, 2825-2830.

[3] Bill Christensen. *Technovelgy.com - Where Science Meets Fiction*. Technovelgy LLC.

[4] Miguel Fernández Zafra. 2019. Text Classification in Python. *Towards Data Science*.

[5] Naive Bayes. 2019. scikit-learn developers. Retrieved from https://scikit-learn.org/stable/modules/naive_bayes.html

[6] Vaishali Ganganwar. 2012. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering* 2, 4 (Apr. 2012), 42-47